Unreliable Heterogeneous Workers in a Pool-based Evolutionary Algorithm

TRACK:

ABSTRACT

Categories and Subject Descriptors

I.2.2 [**Artificial Intelligence**]: Automatic Programming—*program synthesis*

General Terms

Performance,Theory,Experimentation

Keywords

1. INTRODUCTION

Thanks in part to evolutionary computation (EC) research, scientists and engineers from many fields now understand the remarkable power of the natural search process described by the biological theory of Neo-Darwinian evolution [13]. Inspired in biological evolution, EC researchers have developed a variety of search and optimization algorithms [5].

While EAs are inspired by evolution, they mostly follow an abstract model of the natural process. For instance, one aspect that is omitted from most EAs is an open-ended search process, in practice EAs are used to solve problems with well defined objectives, while natural evolution is an adaptive process without an a priori goal or purpose. Such open-ended EAs have been developed, mostly for artistic design [10], interactive evolution [20] and artificial life systems [17]. Other interesting features of biological evolution, is that it is an intrinsically parallel, distributed and asynchronous process, undoubtedly important features that have allowed evolution to produce impressive results throughout nature. However, some of these features are not trivially included into standard EAs, which are mostly coded as sequential and synchronous algorithms [5].

For instance, a large body of work exists in EA parallelization, using multiple CPU cores, multiple nodes and GPUs []. However, distributed and asynchronous EAs have started to become common only recently. In particular, recent trends in information technology have opened new lines of future development for EC research. Today, computing resources computing power range from personal computers and smart-devices to massive data centers. These resources are easily accessible through popular internet technologies, such as cloud computing, peer-to-peer (P2P), and web environments.

Moreover, these technologies are intended for the development of parallel, distributed and asynchronous systems, such that an EA developed on top of them could easily reap the benefits of these features. As stated before, several EAs have been proposed that distribute the evolutionary process among heterogeneous devices, not only among controlled nodes within an in-house cluster or grid, but also to others outside the data center, in users' web browsers, smart phones or external cloud based virtual machines. This reach out approach allows researchers to use low cost computational power that would not be available otherwise, but on the other hand, have the challenge to manage heterogeneous unreliable computing resources. In particular, we are interested in systems that follow a pool-based approach, where the search process is conducted by a collection, of possibly heterogeneous, collaborating processes using a shared repository or population pool. We will refer to succh algorithms as Pool-based EAs or PEAs, and highlight the fact that such systems are intrinsically parallel, distributed and asynchronous.

Despite promising results, PEAs present several notable challenges. From a technological perspective, for example, lost connections, low bandwidth, abandoned work, security and privacy are all important issues that must be addressed. However, here we focus on a common issue with most EAs, that is only amplified in a PEA, a problem that can be referred to as parametrization. In general, among machine learning methodologies, EAs are highly criticized by the large number of parameters they posses, that for real world problems need to be tuned empirically or require additional heuristic processes to be included into the search to adjust the parameters automatically [15]. In the case of a PEAs, this issue is magnified since the underlying system architecture adds several degrees of freedom to the search process.

This work studies a recently proposed pool-based system called EvoSpace, a framework to develop PEAs using a heterogeneous collection of possibly unreliable resources. EvoSpace is based on Linda's tuple space coordination model, where each node asynchronously pulls its work from a central shared memory [11, 10, 9, 23]. The core elements of EvoSpace are a central repository for the evolving population and remote clients here called EvoWorkers which pull random samples of the population to perform on them the basic evolutionary processes (selection, variation and survival), once the work is done, the modified sample is pushed back to the central population. Despite some promising initial results [11, 10, 9, 23], research devoted to EvoSpace has not addressed the parametrization issue mentioned above. In the study presented here, a recent approach called Randomized Parameter Setting Strategy (RPSS) [12, 22] is applied to EvoSpace and tested on several benchmark problems. The idea behind RPSS is that in a distributed EA, algorithm parametrization may be completely skipped for a successful search, with research showing that when the number of distributed process is large enough, algorithm parameters can be set randomly and still achieve good overall results. However, work on RPSS has only focused on the well-known Island Model for EAs, a distributed but synchronous system. On the other hand, the goal of this work is to evaluate RPSS on a complete PEA implemented through EvoSpace.

The remainder of the paper proceeds as follows. Section 2

2. RELATED WORK

Given the computational cost of many EAs, researchers have turned towards internet and cloud based software systems. For instance, Fernández et al. [8]use the well-known Berkeley Open Infrastructure for Network Computing (BOINC) to distribute EA runs across a heterogeneous network of volunteer computers using virtual machines.

However, such a system, as well as others distributed or parallel systems, does not follow the PEA approach studied in the current paper. In general, a pool-based system employs a central repository (real or virtual) where the evolving population is stored. Distributed clients interact with the pool, performing some or all of the basic EA processes (selection, genetic operators, survival). For example, Smaoui et al. [7] uses BOINC redundancy to deal with the volatility of nodes and unreliability of their results. In this work each BOINC work unit consisted of a fitness evaluation task and multiple replicas were produced and sent to different clients, later when outputs were received a validation step ensured all outputs match.

Merelo et al. [16] developed a Javascript PEA that distributes the evolutionary process over the web, this provides the added advantage of not requiring the installation of additional software in each computing node. In this work the server receives an Ajax request with the best individual obtained from the local evolution in clients, and then responds with additional parameters and the best individual in the population so far. Other similar cloud-based solutions are based on a global queue of tasks and a Map-Reduce implementation which normally handles failures by the re-execution of tasks [6, 4, 19].

In general, this pool-based approach can be traced back to the A-Teams system [21], which is not restricted to evolutionary algorithms. Another proposal is made by G. Roy et al. [18], who developed a multi-threaded system with a shared memory architecture that is executed within a distributed environment and achieves substantial performance gains compared to standard approaches. On the other hand, Bollin and Piastra [1] emphasize persistence over performance, proposing a system that decouples population storage from the basic evolutionary operations. A similar decoupled model is proposed by Merelo et al. [**?**], who used a database to store the population that is accessed through a web-server. Finally, a recent PEA is SofEA proposed by Merelo et al. [**?**, **?**], an EA that is mapped to a central CouchDB object store. It provides an asynchronous and distributed search process, where the four main evolutionary operators are decoupled from the evolving population.

This work, however, focuses on EvoSpace, a framework to develop PEAs using heterogeneous and unreliable resources [11, 10, 9, 23], described in greater detail in the following section.

3. EVOSPACE

EvoSpace is based on Linda's tuple space [11] coordination model, where each node asynchronously pulls its work from a central shared memory. The core elements of EvoSpace are a central repository for the evolving population and remote clients called EvoWorkers, which pull random samples of the population to perform on them the basic evolutionary processes (selection, variation and survival), once the work is done, the modified sample is pushed back to the central population.

This model contrasts with the use of a global queue of tasks and implementations of map-reduce algorithms, favored in other proposals PEAs [6, 4, 19]. Following the tuple space model, when individuals are pulled from the EvoSpace container these are removed from it, so that no other EvoWorker can use them. This design decision has several known benefits relevant to concurrency control in distributed systems, and also is an effective way of distributing the workload. Leaving a copy of the individual in the population server free to be pulled by other EvoWorkers will result in redundant work and this could be costly if the task at hand is time consuming. EvoWorkers are expected to be unreliable, as they can loose a connection or are simply shut down or removed from the client. When an EvoWorker is lost, so are the individuals pulled from the repository. Depending on the type of algorithm been executed, the lost of these samples could have a high cost. To address the problem of unreliable EvoWorkers, EvoSpace uses a simple re-insertion algorithm that also prevents the starvation of the population pool. Other pool based algorithms normally use a random insertion technique, but we argue this could negatively impact the outcome of the algorithm in some cases.

EvoSpace [*references hidden for blind review*] consists of two main components (see figure 1): (i) the EvoSpace container that stores the evolving population and (ii) EvoWorkers, which execute the actual evolutionary process, while EvoSpace acts only as a population repository. In a basic configuration, EvoWorkers pull a small random subset of the population, and use it as the initial population for a local EA executed on the client machine. Afterwards, the evolved population from each EvoWorker is returned to the EvoSpace container. When individuals are pulled from the container they remain in a phantom state, they cannot be pulled again but they are not deleted. Only if and when the EvoWorker returns the replacement sample phantoms are truly deleted. If the EvoSpace container is at risk of starvation or optionally when a time-out occurs new phantom individuals are re-inserted to the population and available again. This can be done because a copy of each sample is stored in a priority queue used by EvoSpace to re-insert the sample to the central population; similar to games where characters are Respawned after a certain time. In the experiments conducted in this work re-insertion occurs when the population size is below a certain threshold. Figure 1 illustrates the main components of EvoSpace.

3.1 Implementation

The population of an EA is stored in-memory using the key-value database Redis, chosen over a relational database system, or other non-SQL alternatives, because it provides a hash based implementation of sets and queues which are natural data structures for a PEA model such as EvoSpace. EvoSpace is implemented as a python module and exposed as a web service using the Cherrypy http library. The EvoSpace modules are freely available with a Simplified BSD License from `anonymousURL`.

The EvoSpace system is deployed using Heroku, a multi-language Platfor-as-a-Service (PaaS). The basic unit of composition on Heroku is a lightweight container running a single user-specified process. These containers, which they call *dynos*, can include web (only these can receive `http` requests) and worker processes (including systems used for database and queuing, for instance). Once deployed the web process can be scaled up by assigning more dynos; in our case and in the more demanding configurations of our experiments, the web process was scaled to 20 dynos. Instructions and code for deployment is available at `anonymousURL`. Finally, for the experiments carried out for this paper, EvoSpace workers are distributed using the PiClouf PaaS.

4. PROBLEM STATEMENT

As stated before, one of the main problems in EC research is referred to as parameter tuning. This problem is of particular importance in real-world scenarios, where there can be little prior insights regarding what might be the best configuration for an EA tool, especially if the intent is to use it as a black-box optimizer. Indeed, this problem has received a growing level of interest in recent years, as evidenced by tracks such as Self-Search at GECCO, where the aim is to develop self-adaptive or auto-tuning systems, that reduces the amount of human intervention that might be required before performing an EA-based search.

However, recent work in [12, 22] has shown a promising alternate approach for EAs that employ multiple evolving populations. In particular, the RPSS approach proposed in [12] is as interesting as it is simple. First, consider the original configuration studied in [12, 22], an Island Model EA. In an Island Model, a set of N separate populations, or demes, run semi-isolated evolutionary processes, organized using a particular neighborhood structure, such as ring or a random graph. Each deme is not totally isolated, since after a certain amount of time (generations or function evaluation) a set of individuals is exchanged between neighboring dems, a process known as migration. Obviously, the problem of setting parameter values is magnified by a factor of N, since it is not possible to assume that for a particular problem the best configuration is an homogeneous system where all demes share the same set of parameters. Moreover, the additional complexity of the Island Model incorporates additional degrees of freedom that must be tuned before performing a run.

Such a tuning task can become overwhelming, particularly if the number of islands is excessively large (hundreds).

Table 1: Valid ranges for each EvoWorker parameter.

Parameter	Range
Crossover probability	
Mutation probability	
Tournament size	
Sample Size	
Generations	

Therefore, the proposal in [12] is to set the parameter values randomly, without a tuning or self-adaptive process whatsoever. The RPSS approach is to set the parameters of each deme randomly at the beginning of the run, a very simple and apparently naive approach. Nevertheless, results reported in [12, 22] exhibit promise, achieving competitive results while substantially reducing the amount of effort required to tune the system (the approach only requires the user to specify a range of valid values for each parameter).

Following [12], the proposal of the current work is to apply the RPSS approach to a PEA developed with EvoSpace. However, before turning to the experimental work, lets highlight the main differences between a PEA and the Island Model studied in [12, 22]. First, the Island Model is a synchronous EA, while it implements a higher level of parallelization than a normal EA, and is amenable to distributed implementations, it still relies on a synchronized system in order to perform migration events. Second, the PEA approach based on EvoWorkers can be implemented with as a more heterogeneous system than the Island Model. In particular, the sample (population) size and number of generations executed by each EvoWorker can be different, particularly since there is no need for synchronized migrations. Notice that in EvoSpace, there is no explicit migration process, on the other hand EvoWorkers exchange population members through the centralized pool. These differences could be important regarding the applicability of RPSS on an EvoSpace PEA, which is evaluated in the experimental work presented next.

5. EXPERIMENTAL WORK

The goal of this paper is to evaluate RPSS as a parametrization approach for a PEA developed over EvoSpace, in particular a genetic algorithm (GA). As stated before, the goal is to determine if a random configuration for each of the n EvoWorkers that collaborate on a given run can achieve competitive results compared to a conventional tuning approach. The parameters considered are: 1) crossover probability; 2) mutation probability; 3) tournament size (selection); 4) sample size; and 5) number of generations. The valid ranges established for each parameter are summarized in Table 1.

To gauge the effectiveness of RPSS on a PEA, it is compared with three different parametrization strategies, similar to what is done in [12, 22]. First, the baseline or control method, is an ad-hoc-homogeneous-parametrization (AHP), where all workers share the same homogeneous parameters, set to previously hand-tuned values [11, 10, 9, 23]. Second, the best-homogeneous-parametrization (BHP), where 200 random parametrizations are generated, shared across all EvoWorkers, and the best one is chosen. Finally, the random-heterogeneous-parametrization (RHP), where the

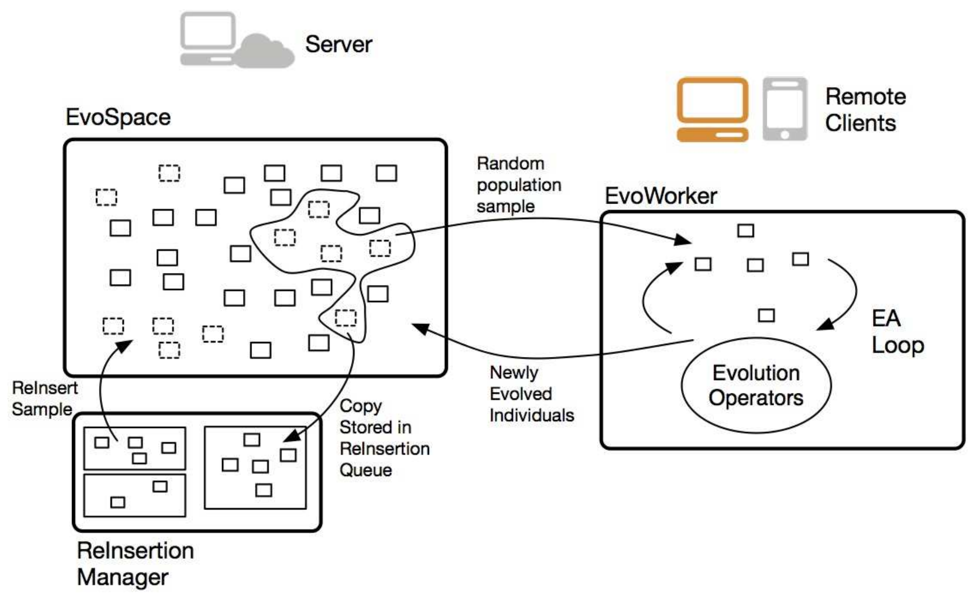

Figure 1: Main components and dataflow within EvoSpace.

parameters of each worker are set independently at random at the beginning of each run. Note that Of all three methods, RHP is the only one that does not require any configuration or parameter tuning before a run.

5.1 Benchmark

The algorithms are evaluated using four common benchmarks. First, the P-Peaks generator of multimodal problems proposed by De Jong et al. [2]. A P-Peaks instance is created by generating a set of P random N-bit strings, which represent the location of the P peaks in the space. To evaluate an arbitrary bit string **x** first locate the nearest peak (in Hamming space). Then the fitness of the bit string is the number of bits the string has in common with that nearest peak, divided by N. The optimum fitness for an individual is 1. This particular problem generator is a generalization of the P-peak problems introduced in [3].

$$f_{P-PEAKS}(\mathbf{x}) = \frac{1}{N} \max_{i=1}^{P} \{N - hamming(\mathbf{x}, Peak_i)\} \quad (1)$$

A large number of peaks induce a time-consuming algorithm, since evaluating every string is computationally hard; this is convenient since to evaluate these type of distributed evolutionary algorithms fitness computation has to be significant with respect to network latency (otherwise, it would always be faster to have a single-processor version). However according to Kennedy and Spears [14] the length of the string being optimized has a greater effect in determining how easy or hard is the problem. In their experiments an instance having P = 200 peaks and N = 100 bits per string is considered to produce a considerably difficult problem.

Then, following [12], three standard real-valued optimization problems are used, these are the Schwefel, Rastrigin, and Griewangk functions, with the number of dimensions set to 40 in each case. For the P-Peaks problem a standard bitstring representation is used, while the other three problems use a real-valued vector representation. Other GA specific details are summarized in Table 2.

Table 2: GA configuration for each benchmark problem.

Feature	P-Peaks	Real-valued problems
Crossover type		
Mutation type		
Survival		
Variable range	$\{0, 1\}$	
Individuals in the Pool		

5.2 Experimental Set-up and Results

Experiments are carried out using a different number of EvoWorkers N on each problem. The first group of runs are done with $N =$ EvoWorkers, and then with $N =$. In each case, and for each problem, 30 independent runs are carried out. Results are summarized by tracking how the best solution found so far varies with respect to the total number of samples taken from the EvoSpace pool of individuals. These results are presented in Figures **??**, where the average +/1 standard deviation of the 30 runs are plotted for each of the three methods evaluated here (AHP, BHP and RHP).

6. CONCLUSIONS AND FURTHER WORK

7. ACKNOWLEDGEMENTS

Hidden for double-blind review.

8. REFERENCES

[1] A. Bollini and M. Piastra. Distributed and persistent evolutionary algorithms: A design pattern. In *Proceedings of the Second European Workshop on Genetic Programming*, pages 173–183, London, UK, UK, 1999. Springer-Verlag.
[2] K. A. De Jong, M. A. Potter, and W. M. Spears. Using problem generators to explore the effects of epistasis. In T. BÃd'ck, editor, *ICGA*, pages 338–345. Morgan Kaufmann, 1997.

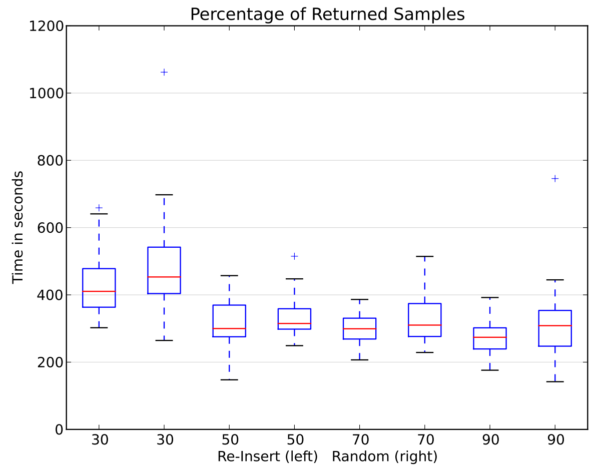

Figure 2: Time required to solution, 4 Workers

[3] K. A. De Jong and W. M. Spears. An analysis of the interacting roles of population size and crossover in genetic algorithms. In *Proceedings of the 1st Workshop on Parallel Problem Solving from Nature*, PPSN I, pages 38–47, London, UK, UK, 1991. Springer-Verlag.

[4] S. Di Martino, F. Ferrucci, V. Maggio, and F. Sarro. Towards migrating genetic algorithms for test data generation to the cloud. In *Software Testing in the Cloud: Perspectives on an Emerging Discipline.*, pages 113–135. IGI Global, IGI Global, 2013.

[5] A. E. Eiben and J. E. Smith. *Introduction to Evolutionary Computing.* SpringerVerlag, 2003.

[6] P. Fazenda, J. McDermott, and U.-M. OâĂŹReilly. A library to run evolutionary algorithms in the cloud using mapreduce. *Applications of Evolutionary Computation*, pages 416–425, 2012.

[7] M. S. Feki, V. H. Nguyen, and M. Garbey. Parallel genetic algorithm implementation for boinc. In B. M. Chapman, F. Desprez, G. R. Joubert, A. Lichnewsky, F. J. Peters, and T. Priol, editors, *PARCO*, volume 19 of *Advances in Parallel Computing*, pages 212–219. IOS Press, 2009.

[8] F. Fernández De Vega, G. Olague, L. Trujillo, and D. Lombraña González. Customizable execution environments for evolutionary computation using boinc + virtualization. 12(2):163–177, 2013.

[9] M. Garcia-Valdez, A. Mancilla, L. Trujillo, J.-J. Merelo, and F. Fernandez-de Vega. Is there a free lunch for cloud-based evolutionary algorithms? In *Evolutionary Computation (CEC), 2013 IEEE Congress on*, pages 1255–1262, 2013.

[10] M. Garcia-Valdez, L. Trujillo, F. Fernández de Vega, J. Merelo Guervós, and G. Olague. EvoSpace-Interactive: A Framework to Develop Distributed Collaborative-Interactive Evolutionary Algorithms for Artistic Design. In J. C. A. Machado, Penousal; McDermott, editor, *Evolutionary and Biologically Inspired Music, Sound, Art and Design*, volume 7834 of *Lecture Notes in Computer Science*, pages 121–130. Springer Berlin Heidelberg, 2013.

[11] M. García-Valdez, L. Trujillo, F. Fernández de Vega, J. J. Merelo Guervós, and G. Olague. EvoSpace: A Distributed Evolutionary Platform Based on the Tuple Space Model. In A. Esparcia-Alcázar, editor, *Applications of Evolutionary Computation*, volume 7835 of *Lecture Notes in Computer Science*, pages 499–508. Springer Berlin Heidelberg, 2013.

[12] Y. Gong and A. Fukunaga. Distributed island-model genetic algorithms using heterogeneous parameter settings. In *IEEE Congress on Evolutionary Computation*, pages 820–827. IEEE, 2011.

[13] J. H. Holland. *Adaptation in Natural and Artificial Systems.* MIT Press, Cambridge, MA, USA, 1992.

[14] J. Kennedy and W. Spears. Matching algorithms to problems: an experimental test of the particle swarm and some genetic algorithms on the multimodal problem generator. In *Evolutionary Computation Proceedings, 1998. IEEE World Congress on Computational Intelligence., The 1998 IEEE International Conference on*, pages 78–83, May.

[15] O. Kramer. *Self-Adaptive Heuristics for Evolutionary Computation*, volume 147 of *Studies in Computational Intelligence.* Springer, 2008.

[16] J. Merelo-Guervos, P. Castillo, J. L. J. Laredo, A. Mora Garcia, and A. Prieto. Asynchronous distributed genetic algorithms with Javascript and JSON. In *Evolutionary Computation, 2008. CEC 2008. (IEEE World Congress on Computational Intelligence). IEEE Congress on*, pages 1372–1379, June.

[17] C. Ofria and C. O. Wilke. Avida: A software platform for research in computational evolutionary biology. *Artif. Life*, 10(2):191–229, 2004.

[18] G. Roy, H. Lee, J. L. Welch, Y. Zhao, V. Pandey, and D. Thurston. A distributed pool architecture for genetic algorithms. In *Proceedings of the Eleventh conference on Congress on Evolutionary Computation*, CEC'09, pages 1177–1184, Piscataway, NJ, USA, 2009. IEEE Press.

[19] D. Sherry, K. Veeramachaneni, J. McDermott, and U.-M. OâĂŹReilly. Flex-gp: Genetic programming on the cloud. In C. Chio, A. Agapitos, S. Cagnoni, C. Cotta, F. Vega, G. Caro, R. Drechsler, A. EkÃąrt, A. Esparcia-AlcÃązar, M. Farooq, W. Langdon, J. Merelo-GuervÃşs, M. Preuss, H. Richter, S. Silva, A. SimÃţes, G. Squillero, E. Tarantino, A. Tettamanzi, J. Togelius, N. Urquhart, A. Uyar, and G. Yannakakis, editors, *Applications of Evolutionary Computation*, volume 7248 of *Lecture Notes in Computer Science*, pages 477–486. Springer Berlin Heidelberg, 2012.

[20] H. Takagi. Interactive evolutionary computation: fusion of the capabilities of ec optimization and human evaluation. *Proceedings of IEEE*, 89(9):1275–1296, 2001.

[21] S. Talukdar, L. Baerentzen, A. Gove, and P. De Souza. Asynchronous teams: Cooperation schemes for autonomous agents. *Journal of Heuristics*, 4(4):295–321, 1998.

[22] R. Tanabe and A. Fukunaga. Evaluation of a randomized parameter setting strategy for island-model evolutionary algorithms. In *IEEE Congress on Evolutionary Computation*, pages 1263–1270. IEEE, 2013.

[23] L. Trujillo, M. G. Valdez, F. F. de Vega, and J. J. M. GuervÃşs. Fireworks: Evolutionary art project based on evospace-interactive. In *IEEE Congress on Evolutionary Computation*, pages 2871–2878. IEEE, 2013.